ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Machine Learning for Finance

Applying Statistical Modeling and Deep Learning to Perform Stock Price Forecasting

Authors:
Ridha Chahed
Gerald Sula

Supervisors:
Dr. Damien Ackerer

Lausanne, January 2021

Abstract

Prediction and analysis of stock market data have an important role in today's economy. The various algorithms used for forecasting can be categorized into linear (AR, MA, ARIMA, ARMA) and non-linear models (Neural Network, RNN, LSTM). In this paper, we aim to benchmark those different models using day-wise closing prices of 20 companies of the Dow Jones index from 2010 to 2020. We perform various statistical tests on the data in order to decide on the infrastructure of our pipeline, as well as perform some feature augmentation specific to financial data.

Table of contents

1 Introduction

When it comes to trading, there are fundamentally two main methods of analysis related to stock prediction: Fundamental Analysis and Quantitative Analysis. For the first one, trading decisions are based on publicly available information regarding the market value, statistics on a company or financial statements. Whereas the second method, Quantitative Trading, takes advantage of advanced mathematical models, to make objective predictions regarding the financial market. Today, it is also possible to combine the two, thanks to advanced Natural Processing techniques, but this goes beyond the scope of this project.

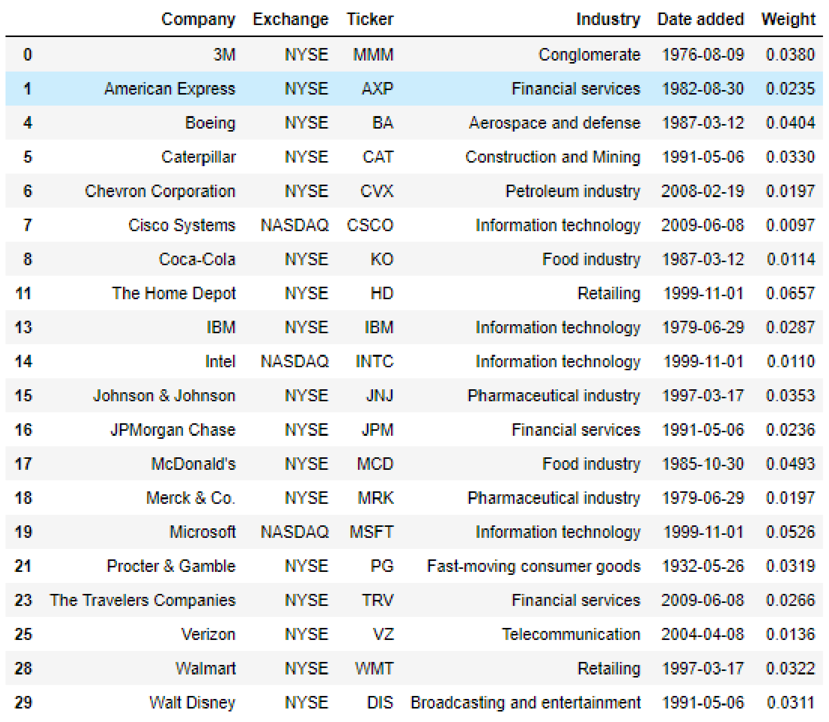
We focus on quantitative analysis of stock prices from the Dow Jones, using state of the art deep learning techniques. Namely, we experiment with Long Short-Term Models (LSTM), as well as Recurrent Neural Networks models (RNN), which have been proven to be very effective when it comes to highly volatile time series, such as the stock price. As a baseline to these deep learning models we use a simple linear regression model such as Auto Regressive Integrated Moving Average (ARIMA).

We use data regarding the stock price as input to these models, as well as some additional financial indicators that are computed on the data, then we measure the accuracy of the predictions.

2 Dataset and Features

2.1 Data

We are using stock information related to the 30 companies of the Dow Jones Index (see Figure 1). The data is scrapped from Google Finance and contains information about the different financial factors of each company such as close price, open price,high, low, trading volume etc. Since the list of companies included in the Dow Jones index has been evolving through the years, with certain companies being removed from the index, and new ones added, we have decided to only look at companies which have been present from 2010 to 2020, which leaves us with 20 companies. We decide to set as the target variable the daily adjusted close price.



	Company	Exchange	Ticker	Industry	Date added	Weight
0	3M	NYSE	MMM	Conglomerate	1976-08-09	0.0380
1	American Express	NYSE	AXP	Financial services	1982-08-30	0.0235
4	Boeing	NYSE	BA	Aerospace and defense	1987-03-12	0.0404
5	Caterpillar	NYSE	CAT	Construction and Mining	1991-05-06	0.0330
6	Chevron Corporation	NYSE	CVX	Petroleum industry	2008-02-19	0.0197
7	Cisco Systems	NASDAQ	CSCO	Information technology	2009-06-08	0.0097
8	Coca-Cola	NYSE	KO	Food industry	1987-03-12	0.0114
11	The Home Depot	NYSE	HD	Retailing	1999-11-01	0.0657
13	IBM	NYSE	IBM	Information technology	1979-06-29	0.0287
14	Intel	NASDAQ	INTC	Information technology	1999-11-01	0.0110
15	Johnson & Johnson	NYSE	JNJ	Pharmaceutical industry	1997-03-17	0.0353
16	JPMorgan Chase	NYSE	JPM	Financial services	1991-05-06	0.0236
17	McDonald's	NYSE	MCD	Food industry	1985-10-30	0.0493
18	Merck & Co.	NYSE	MRK	Pharmaceutical industry	1979-06-29	0.0197
19	Microsoft	NASDAQ	MSFT	Information technology	1999-11-01	0.0526
21	Procter & Gamble	NYSE	PG	Fast-moving consumer goods	1932-05-26	0.0319
23	The Travelers Companies	NYSE	TRV	Financial services	2009-06-08	0.0266
25	Verizon	NYSE	VZ	Telecommunication	2004-04-08	0.0136
28	Walmart	NYSE	WMT	Retailing	1997-03-17	0.0322
29	Walt Disney	NYSE	DIS	Broadcasting and entertainment	1991-05-06	0.0311

Figure 1: List of the companies listed in the Dow Jones index

2.2 Features

Along with the basic stock information we obtained when scraping the web, we are performing some feature augmentation to include some technical indicators that help in observing momentum, volatility and trends.

- Simple Moving average
- Cumulative moving average
- Exponential moving average
- Bollinger Bands

2.3 Data scaling

Additionally, all the data are scaled before training, in order to make training less sensitive to the original scale of the different features and as well as to make sure that regularization behaves similarly for features of different scales. We have tested scaling the data on a scale of [0,1], as well as [-1,1] and observed that the latter performs better.

3 Methods

In order to observe the performance of the different methods, we have built 3 models and compared the results. For benchmarking, we are using an ARIMA model, and two complex deep learning models LSTM and RNN.

Since we are dealing with time series, we have sequentially split the data into test and train sets. For training, we are using data from 2010 to 2018, and for testing we keep only data from 2019, so we end up with a 90/10 split.

The metric we are using to evaluate the models is the Root Mean Square Error, that we will compare between the actual price in the test set and the prediction we got from each model.

Out of the 20 available datasets we are using one of them for training and hyperparameter optimization. Then we use the best performing model, together with the best parameters, to train and test on the remaining sets and see if it performs just as well on the others. The dataset we have picked for hyperparameter tuning is American Express Company (AXP).

3.1 Autoregressive integrated moving average (ARIMA)

Autoregressive model (AR) specifies that the output depends linearly on its previous values and on a stochastic element.
AR(p) model is an autoregressive model of order p :

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t$$

with the φ_i the model parameters, c a constant and ϵ_t a white noise.

Moving average model (MA) specifies that the output depends linearly on the current and various past values of a stochastic element.
MA(q) refers to the moving average model of order q :

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

with μ the mean of the series, θ_q the model parameters and ϵ_t the white noise elements.

Autoregressive moving average model (ARMA) describes a stationary stochastic process in terms of an autoregression model and a moving average model.
ARMA(p,q) refers to a model with p autoregressive elements and q moving average elements:

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$$

Using the lag operator L we get

$$\left(1 - \sum_{i=1}^{p} \varphi_i L^i\right) X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t$$

Autoregressive integrated moving average (ARIMA) is a generalization of the ARMA model and can be used when the data is non stationary in terms of mean. To fix this issue, an initial differentiation step can be applied several times to remove the non mean stationarity. Therefore, the data values are replaced with the difference between their values and the previous ones.
ARIMA(p,d,q) refers to a model with p autoregressive elements, d degree of differencing and q moving average elements:

$$\left(1 - \sum_{i=1}^{p} \varphi_i L^i\right) (1-L)^d X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t$$

To apply these statistical methods, we make the assumption that the time series is stationary. There is three criteria for a time series to be categorized as stationary :

1. Constant mean over time
2. Constant variance over time also known as the homoscedasticity property
3. Autocovariance function does not depend on time

A stationarized series is relatively easy to predict as we simply predict that its statistical properties will remain the same. However, as we can see in Figure 2 these conditions are not met.

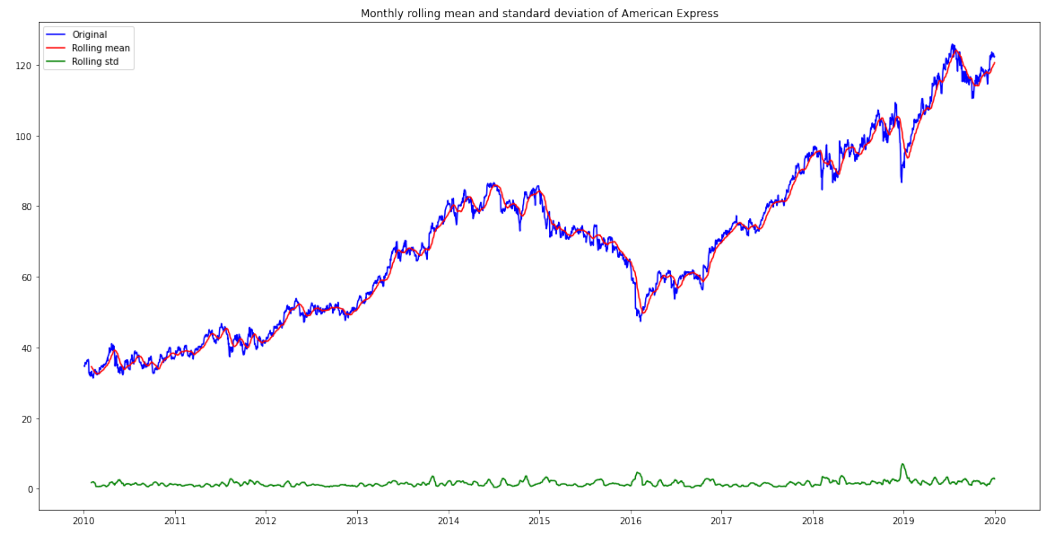


Figure 2: Monthly rolling mean and standard deviation of American Express stock

Throughout several transformations like detrending, differencing and log transform a non stationary time series can be made stationary. We can then use the statistical methods, make predictions on the stationarized series and finally reverse the transformations on the predictions to obtain results for the original data.

After the transformations to test if a time series is stationary we apply the Dickey Fuller test:

$$X_t = \alpha + \rho X_{t-1} + \epsilon_t$$

$$\implies X_t - X_{t-1} = \alpha + (\rho - 1)X_{t-1} + \epsilon_t$$

We test if $\rho - 1$ is significantly different from zero or not. If the null hypothesis is rejected we have a stationary time series. We set the critical value to 0.05.

We apply the Dickey Fuller test on original time series for American Express stock. The test statistic of -0.28 is bigger than the critical value of -2.86 at 5%. This implies that we fail to reject the null hypothesis that the time series is non stationary We obtain the same result for the log transform, we fail to reject the null hypothesis.

We must make the series stationary through other transformations. There can be 2 mains reasons for non stationarity :

1. Trends, with mean varying over time
2. Seasonality with variations at specific time frames

We check the presence of trends and seasonality, if they are present we model them and remove them from the original series. For that we compute the moving average and the exponentially weighted moving average with a window size of 153 days and we use it to detrend the time series and its log transform.

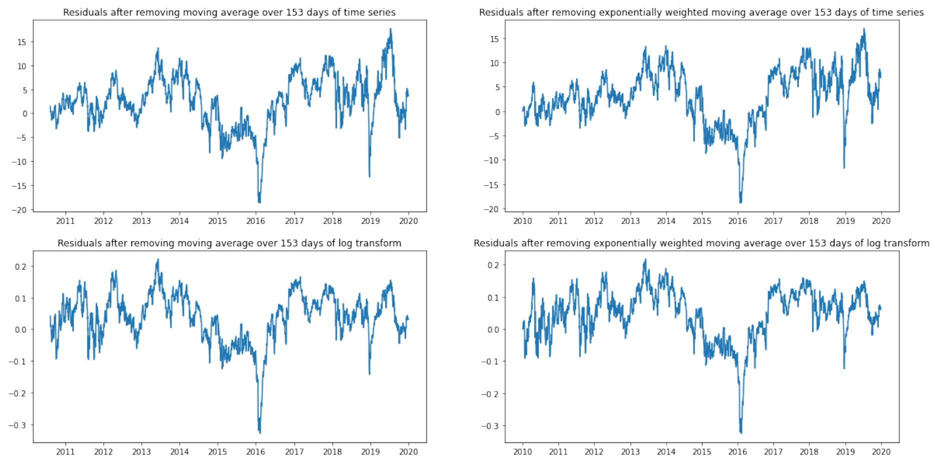


Figure 3: Detrended time series

For all transformations the test statistic is smaller than the 5% critical value and the p-value < 0.05 which allows us to reject the null hypothesis and to claim that the transformed time series is stationary. We also try other transformations such as first order differencing on the original time series.

Once we have a stationary series we need to know:

1. Is it an AR or MA process ?

   The main difference between AR and MA models relies on the correlation function. For MA models the correlation between X_t and X_{t-n} for n superior to order of MA equals zero. On the other hand for the AR models the correlation of X_t and X_{t-n} declines as n grows.

2. What order of AR or MA processes do we need to use ?

   Autocorrelation function (ACF) is a plot of total correlation between different lags of the process. For a MA series of degree n we will not get any correlation between X_t and $Xt–(n+1)$, so it's easy to find the correct lag value n. On the other hand, for the AR series the correlation will gradually go down so we need another method for it. We use the partial autocorrelation function (PACF) which gives us the partial correlation of each lag and excludes the effect of the other ones. We expect to find a cut off after the degree of the AR series. So we use these methods to tune the parameters of our statistical models. If both ACF and PACF decreases gradually it indicates that to stationarize the series by for example differencing adding as parameters the differencing degree d.

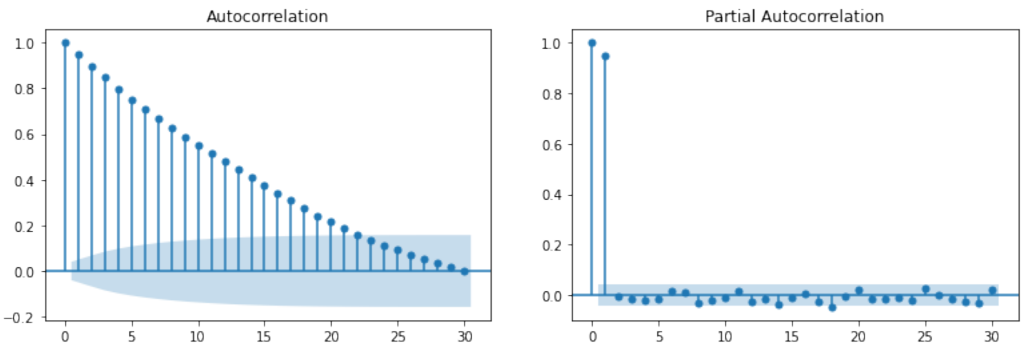


Figure 4: Left: Autocorrelation function. Right: Partial correlation function

3.2 LSTM and RNN

Recurrent Neural Networks (RNN) are neural networks that have the ability to store past information as an internal state. Therefore, as the new information flows in the model it can decide based on it's 'memory'. If we unwrap the recurrent neural network through time it can be thought as a chain of similar networks passing information one to the other.

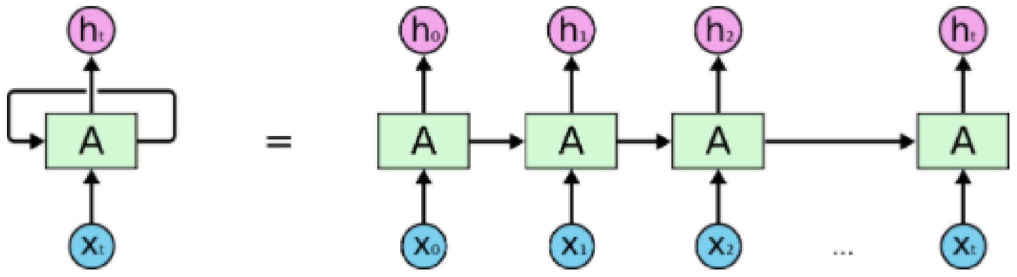
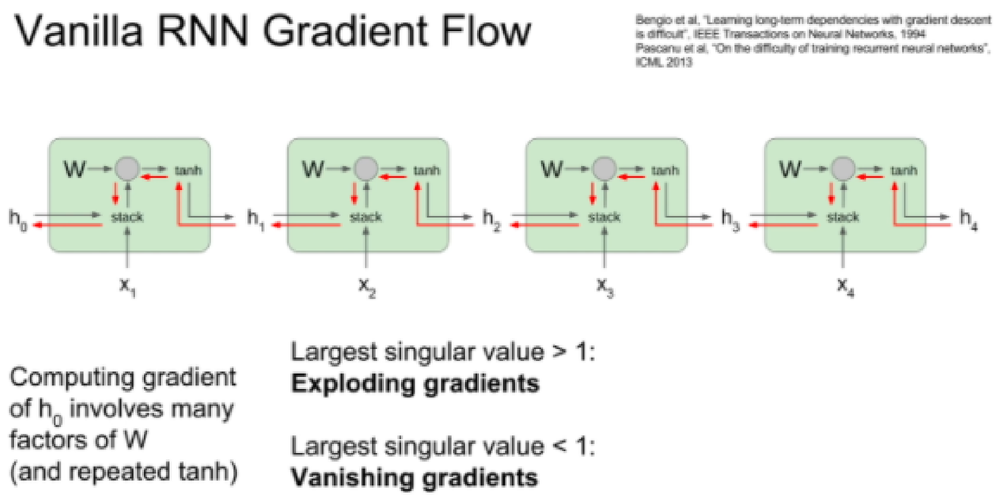


Figure 5: Unrolled RNN

During training, when we compute backpropagation from h_t to h_{t-1} we have a multiplication by $\mathbf{W}^\top$. So computing gradient of h_0 for example involves many factors of $\mathbf{W}^\top$ and repeated tanh.



Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1: **Exploding gradients**

Largest singular value < 1: **Vanishing gradients**

Figure 6: Gradient flow during backpropagation

To tackle the problem of exploding gradients, gradient clipping is used, that is, we scale the gradient if its norm becomes too big.

For the vanishing gradient problem we need to move to a more complicated RNN architecture : **LSTM**. Its design obliviates the problems of vanishing and exploding gradients with an architecture that has better gradient flow properties.

RNN has a hidden state and we use the recurrent relation to update the hidden state at each time step. In LSTM we maintain two hidden states, H_t the hidden state similar to RNN state and also C_t the cell state which is an internal state kept inside the LSTM. The cell state can be seen as a highway where the information can flow without a lot of changes. This state can be modified by throughout gates, LSTM module has three of them.

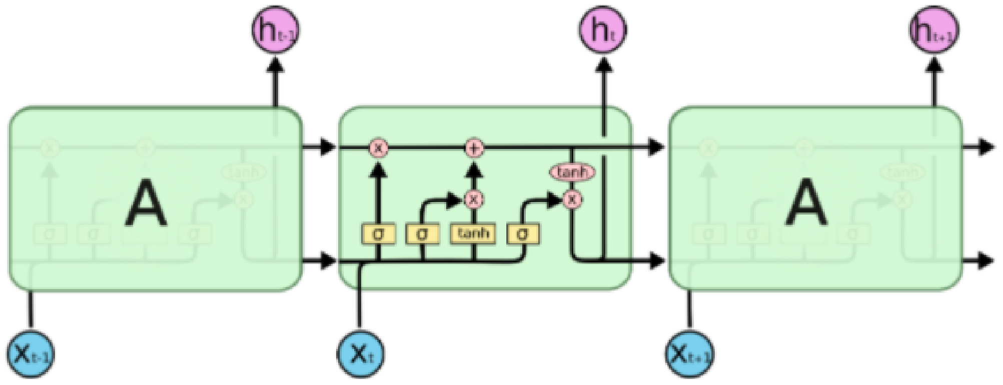


Figure 7: LSTM architecture

1. **Forget gate**: It decides what information of the cell state is kept. It takes as input the past cell state h_{t-1} and the present input X_t and outputs a number between 0 and 1 for each dimension of the cell state. An element wise multiplication is then applied.
2. **Input gate**: It decides what new information is stored in the cell state and it's done in 2 steps. First a sigmoid layer chooses which dimension is updated, then a tanh layer outputs the new values that need to be added to the cell state.

3. **Output gate**: The cell state passes through a tanh layer to have values in [-1,1] and then multiply it by the output of a sigmoid layer to decide what parts of the cell state are output.

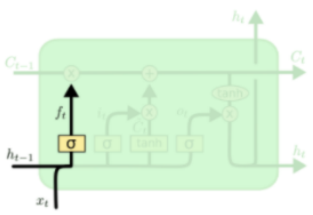


Figure 8: Forget gate

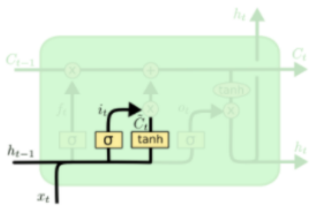


Figure 9: Input gate

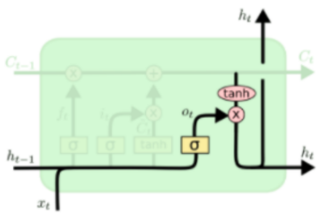


Figure 10: Output gate

4 Results

4.1 ARIMA

Hyperparameter tuning with training set from 2010 to 2017, the cross validation set is the year 2018 and test set is the year 2019. We apply a grid search over the 3 parameters :

- p autoregressive elements with the range [0,4]
- d degree of differencing with the range [0,1]
- q moving average elements with the range [0,3]

The best result is obtained with ARIMA(1, 1, 2) with a RMSE on cross validation of 1.3534 and a RMSE on the test set of 1.1928.

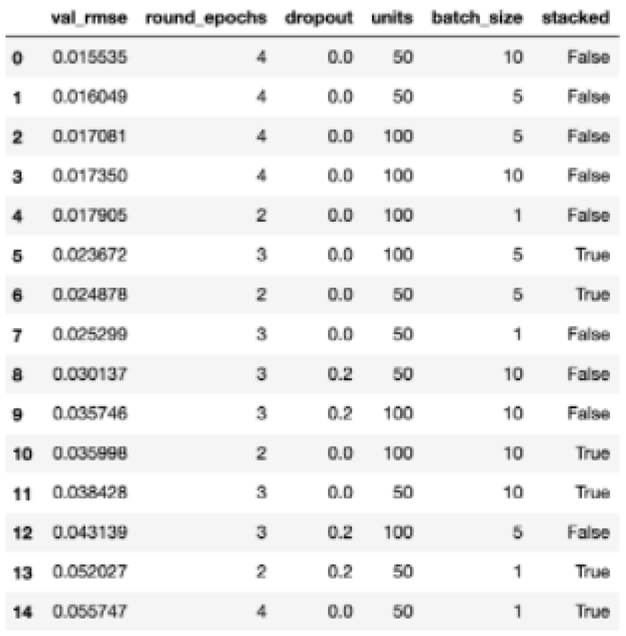
4.2 RNN

In order to achieve the best results using RNN we are adding some of the previous price points to every row of our train set. So we are setting a number for the window size, and we are augmenting our features by including the prices of the past window size - 1 points for every point in time. In addition to that, we are also adding the 4 technical indicators mentioned in the previous sections of the report. So in total, every row of the train/test set now has (window size + 4) columns.

After some trial and error we have decided to set the window size to 20, as there did not seem to be any meaningful improvement if we increased in any further.

We are performing some hyper parameter optimization, in order to find the best combination of parameters. We have tested multiple values for the number of units in the RNN layer, the batch size, the dropout rate and as well as different optimizers for our network. To correctly test the performance of our model, we are using 10% of the train set, as a validation set. For every combination of the model's parameters, we are training on the train set, and evaluating the RMSE score on the validation set. Finally, we save the best model and the best parameters, of the iteration that resulted in the lowest validation score.

We are trying different parameters for the number of units in the RNN layer, stacked/not-stacked layers, batch size, dropout rate and optimizer.



	val_rmse	round_epochs	dropout	units	batch_size	stacked
0	0.015535	4	0.0	50	10	False
1	0.016049	4	0.0	50	5	False
2	0.017081	4	0.0	100	5	False
3	0.017350	4	0.0	100	10	False
4	0.017905	2	0.0	100	1	False
5	0.023672	3	0.0	100	5	True
6	0.024878	2	0.0	50	5	True
7	0.025299	3	0.0	50	1	False
8	0.030137	3	0.2	50	10	False
9	0.035746	3	0.2	100	10	False
10	0.035998	2	0.0	100	10	True
11	0.038428	3	0.0	50	10	True
12	0.043139	3	0.2	100	5	False
13	0.052027	2	0.2	50	1	True
14	0.055747	4	0.0	50	1	True

Figure 11: Top 15 after cross validation of the RNN parameters

The best performing parameters we have found are the following:

Batch size: 10, Dropout: 0.0, Optimizer: Adam, RNN units: 50, Stacked: False

This resulted in a training score of 0.99 RMSE and a testing score of 1.43 (after undoing the scaling of the data).

4.3 LSTM

Similarly to the pipeline used for RNN, we have also prepared a model using LSTM layers. Once again we take as input to the model the windowed prices of the past 20 days, together with the 4 technical indicators mentioned above.

We tested different numbers of layers and shapes for the LSTM model, but it seems that there is a decrease in performance when using more than one LSTM layer. That is why during the hyperparameter tuning phase, we are considering only a single layer. This time the grid search includes different values for dropout rate, recurrent dropout rate, batch size and number of units in the LSTM layer

Once again, we are using 10% of the train set as a validation set that will be used as validation of the performance during the hyper parameter optimization.

Finally the best performing parameters we have found are the following: Batch size:1, Dropout: 0.0, epochs: 4, Recurrent Dropout: 0.0, Optimizer: Adam, LSTM units: 50

This resulted in a train score of 0.97 RMSE, after undoing the scaling originally performed on the data and RMSE score of 1.22 on the test set.

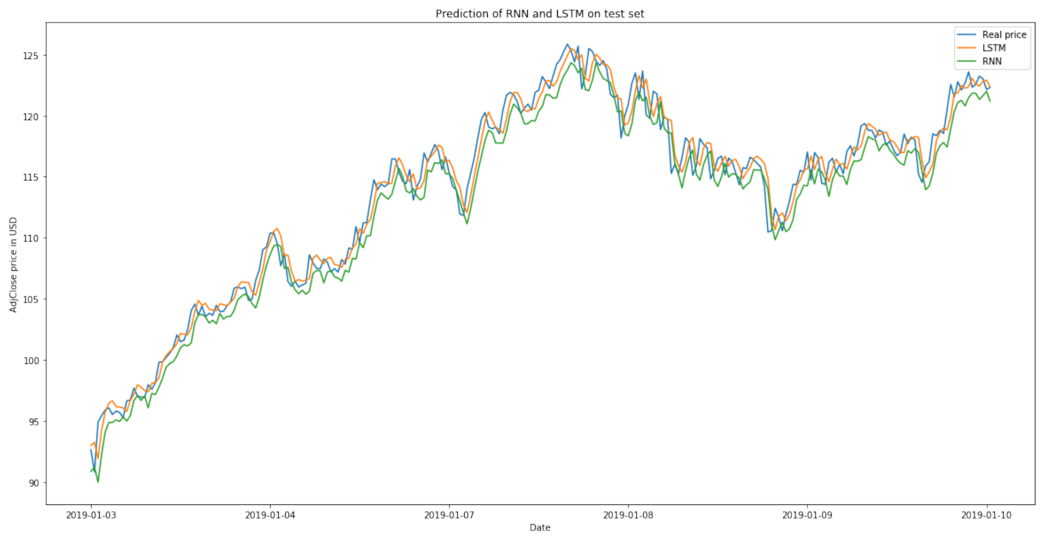
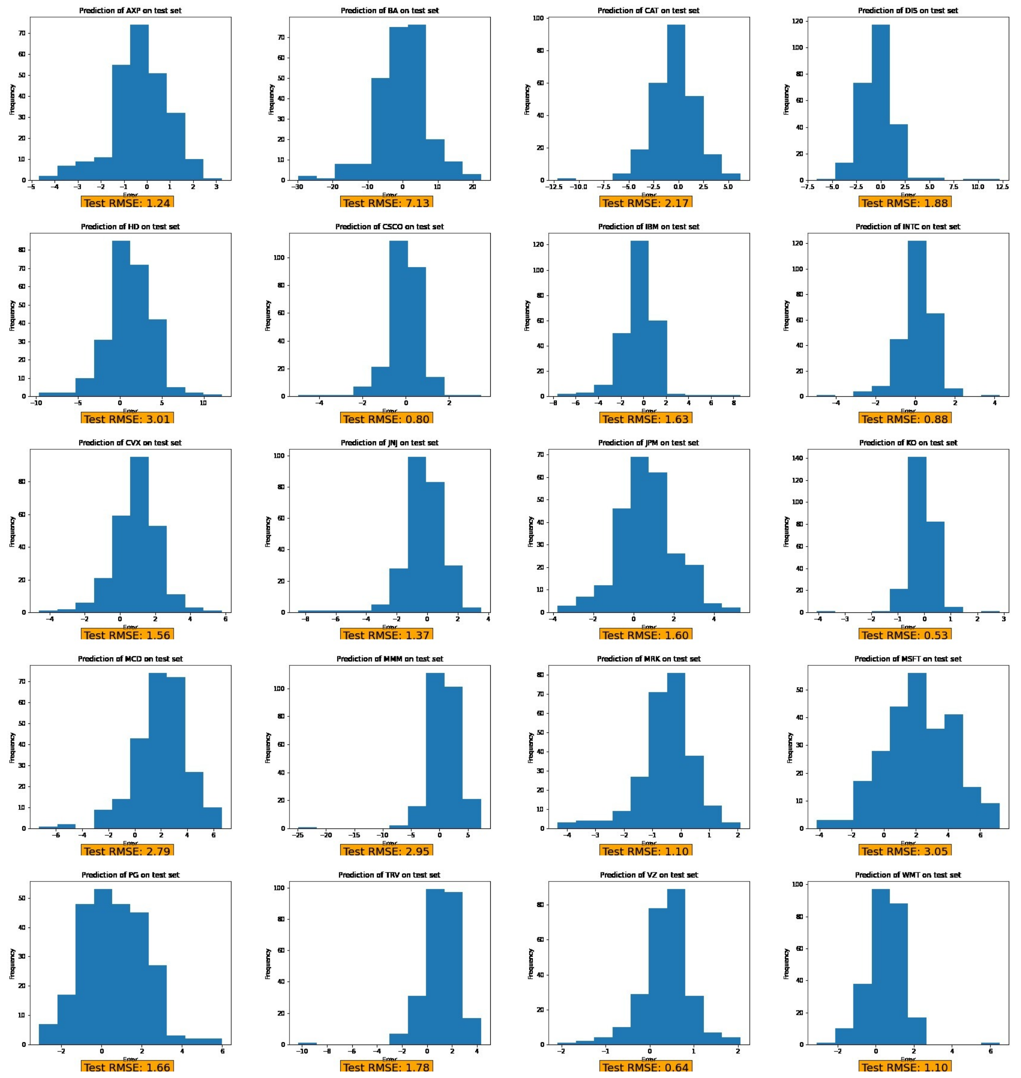


Figure 12: RNN and LSTM predictions on the test set

5 Generalization to DJIA stocks

Finally we are ready to test the performance of the best model we found on the rest of the Dow Jones Stocks. We are repeating the same data transformation as we did for the AXP stock, so scaling it, computing the augmented features together with the window size of 20. Each one of the sets is split into train set (years 2010 to 2018) and test (year 2019). We are building an LSTM model using the best parameters we found, to train each of them separately, and test on the appropriate data. Instead of visualizing the real price alongside with the predicted price, we are instead showing the histograms of the errors in the predictions (that is real price - predicted price) computed for each day of the test set. We are also showing the RMSE score of the whole test set on the bottom of every plot. Here are the results.



As we can see from Figure 14, our model is performing reasonably well in the majority of cases. There are a few exceptionally well performing cases with an RMSE score close to 0.6, but for the most part the RMSE score is close to 1.4.

However there is one exception, the BA stock (Boeing Company) that resulted in a score of 7.32. But if we take a closer look at the data we can see why the model is performing so poorly.

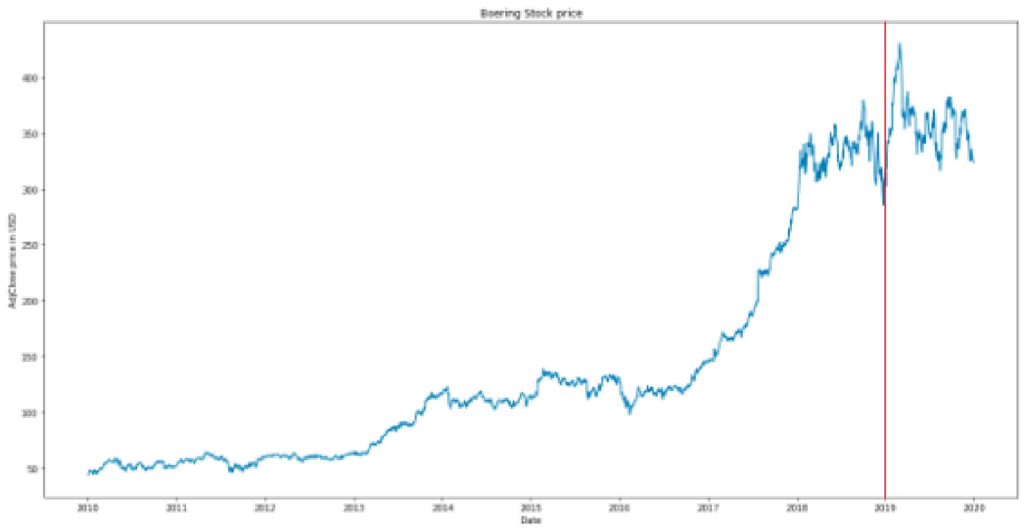


Figure 13: Boeing Company stock evolution

On the training set the price has been steadily growing with some confusion during the last months. While the test set starts with a big increase in price and then proceeds to behave very differently compared to the past. This change of behavior between the train set and the test set can explain the poor performance of the LSTM model.

Figure 14: Distribution of the residuals of the predictions on the test set for each stock

6 Conclusion

In this project we have explored several algorithms to predict stock prices. We have shown that through training and parameter optimization we can achieve very low errors for this regression task. Like the no free lunch theorem states there isn't a model that performs better than all the others on all the stocks. We also have observed that in general the machine learning models perform poorly on stocks whose behaviour changes drastically between the training set and the test set. One way to avoid this problem is to dynamically update the model as the market changes.

References

[1] Stanford University. *Lecture 10: Recurrent Neural Networks.*

[2] Zhichao Zou and Zihao Qu. *Using LSTM in Stock prediction and Quantitative Trading.*

[3] Mikko. *Hyperparameter Optimization with Keras.* Medium article.

[4] Christopher Olah. *Understanding LSTM Networks*

[5] Brian Griner. *Tuning the Beast: Review of LSTM Tuning Methods for Forecasting Time Series*